(12)

CHEMICAL RESEARCH, DEVELOPMENT & ENGINEERING CENTER

CRDEC-TR-356

MEASUREMENT OF A MICHELSON INTERFEROMETER MIRROR VELOCITY WITH A TIME INTERVAL ANALYZER

R.J. Combs
DETECTION DIRECTORATE

R.B. Knapp
RESEARCH DIRECTORATE

C.T. Cathey
TRI-SQUARE CORPORATION
Ormond Beach, FL 32175

July 1992

U.S. ARMY
ARMAMENT
MUNITIONS
CHEMICAL COMMAND

Aberdeen Proving Ground, Maryland 21010-5423

Disclaimer

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorizing documents.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1992 July	Final, 92 Feb - 92 Jun

4. TITLE AND SUBTITLE

Measurement of a Michelson Interferometer Mirror Velocity with a Time Interval Analyzer

5. FUNDING NUMBERS

TA-DARPA 8304

6. AUTHOR(S)

Combs, R.J., Knapp, R.B. (CRDEC), and
Cathey, C.T. (Tri-Square Corporation)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

CDR, CRDEC, ATTN: SMCCR-DDT/SMCCR-RSL-S,
APG, MD 21010-5423

Tri-Square Corporation, P.O. Box 32175,
Ormond Beach, FL 32175

8. PERFORMING ORGANIZATION REPORT NUMBER

CRDEC-TR-356

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report documents the design, construction, and implementation of a time interval analyzer (TIA). The TIA is a precision counter that measures the dynamic variation in the time intervals (i.e., frequency) of a signal. The helium-neon (HeNe) laser signal from the servo mirror control circuitry of the interferometer provides a nearly constant frequency source. The HeNe signal frequency is directly proportional to the interferometer mirror velocity. A constant velocity is crucial for the proper operation of a Michelson interferometer, which is used in a Fourier transform spectrometer. The TIA provides a critical evaluation of velocity variations in a single interferometer mirror scan via analysis of the HeNe laser signal.

14. SUBJECT TERMS

Time interval counter
Time interval analyzer
Fourier transform spectroscopy

15. NUMBER OF PAGES

48

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Blank

PREFACE

The work described in this report was authorized under DARPA Task No. 8304. This work was started in February 1992 and completed in June 1992.

The use of trade names or manufacturers' names in this report does not constitute an official endorsement of any commercial products. This report may not be cited for purposes of advertisement.

This report has been approved for release to the public.

Blank

CONTENTS

LIST OF FIGURES AND TABLES

Figures

Tables

Blank

MEASUREMENT OF A MICHELSON INTERFEROMETER MIRROR VELOCITY WITH A TIME INTERVAL ANALYZER

1. INTRODUCTION

Time interval analyzers (TIA) provide a means to characterize a wide variety of rapidly changing signals.[1] A TIA is used to monitor an indexing helium-neon (HeNe) laser signal from a Michelson interferometer. The indexing HeNe laser signal is available from the servo mirror velocity control circuitry of a Michelson interferometer. Most commercial Fourier transform infrared (FTIR) spectrometers implement the Michelson interferometer design and use an indexing HeNe laser reference in control of mirror velocity.

The Michelson interferometer is constructed using an optical beamsplitter, fixed mirror, and movable mirror (Figure 1). The incoming IR radiation (i.e., 2.5-25 μm wavelength) is divided into two optical paths at the beamsplitter. The beamsplitter is positioned at a 45° angle to the input radiation. Each optical path contains 50% of the total IR energy. One path is to the fixed mirror, while the second optical path is to the movable mirror. After reflection from the fixed and movable mirrors, the IR radiation returns to be recombined at the beamsplitter. This generates an interference pattern. The IR photodetector is exposed to this pattern. The pattern is modulated by the relative positions of the fixed and movable mirrors to the beamsplitter. Monochromatic light generated by the HeNe laser transverses a parallel optical path to the IR radiation. A visible light photodetector receives the modulated monochromatic HeNe laser light (i.e., laser fringes). The photodetector converts the laser fringes into a sinusoidal reference signal. Selected zero crossing of the sinusoidal reference signal allows the IR interference pattern to be sampled at equally spaced and well-defined mirror positions.

Variation in the movable mirror velocity by as little as $\pm 2\%$ can result in a significant mirror position sampling error. This sampling error degrades the signal-to-noise ratio (SNR) of the acquired IR spectrum.[2] For a sufficiently high SNR, the FTIR spectrometer permits identification and quantification of a wide variety of materials. Decrease in the SNR becomes important since this lowers the ability of the spectrometer to detect the material of interest. The purpose of this article is to present a TIA capable of measuring the mirror velocity of a Michelson interferometer and any velocity variations within a single mirror scan.

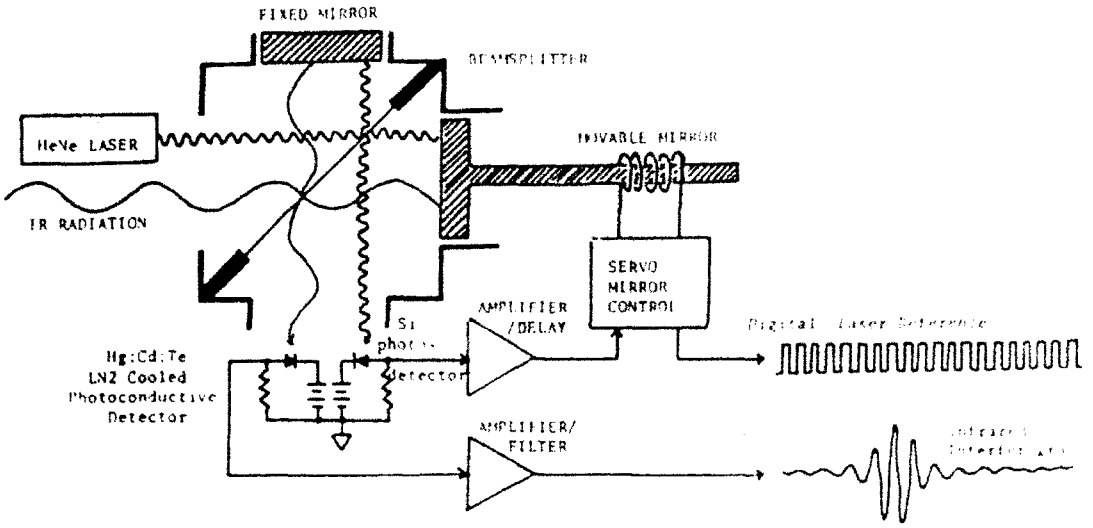

Figure 1. Elements of a Fourier Transform Infrared (FTIR) Spectrometer

2. TIA DESCRIPTION

The TIA is composed of three parts: a time interval counter (TIC) circuit, an interval counter buffer memory (ICBM) circuit, and a program called TICOUNT. The TIC circuit times the intervals between data sampling points. The data sampling points are obtained from selected zero crossing of the HeNe laser reference signal. The ICBM circuit saves the time interval counts in a buffer memory and interfaces that buffer memory to the IBM/PC-XT bus. The TICOUNT program reads and analyzes the data contents of the memory buffer. This analysis determines the average time interval count (tic) and the variations from the average time interval.

2.1 TIC Circuit.

The TIC circuit measures the duration of both the high and low intervals of a digital signal. A comparator converts the sinusoidal laser reference signal into the digital laser reference (DLR) signal (i.e., square wave). Duration of the high or low DLR signal provides a means to measure the mirror velocity over the entire movable mirror displacement in the Michelson interferometer. Acquisition of the interval counts for successive high and low intervals in the DLR signal permits evaluation of the interferometer mirror velocity variation during each mirror scan. For this evaluation, reciprocal counters, which measure only the average period of many cycles, cannot determine the position of velocity errors within a single mirror scan.[3]

The TIC circuit consists of a crystal oscillator-generated time base, digital transition detector, and counters with latched outputs. Figure 2 is a schematic of the TIC circuitry. The crystal oscillator, IC_6A, generates a 67.7576 MHz time base reference. The oscillator drives the clock inputs of the IC_1A data flip flops. The IC_2A flip flops along with the gating of IC_5A permits detection of either a high-to-low or a low-to-high transition of the DLR signal. Detection of a transition is only possible, when the status signal (SSTAT) is an active high. Figure 3 illustrates the operation of the digital transition detector. A low-to-high transition on the DLR signal line produces an active low pulse on the $\overline{\text{LEAD}}$ signal line, while a high-to-low DLR signal transition gives an active low pulse on the $\overline{\text{TRAIL}}$ signal line. The $\overline{\text{HI}}$ and $\overline{\text{LOW}}$ outputs of IC_2A, which track the DLR signal are toggled by the $\overline{\text{LEAD}}$ and $\overline{\text{TRAIL}}$ signals. The signals $\overline{\text{HI}}$, $\overline{\text{LOW}}$, $\overline{\text{LEAD}}$ and $\overline{\text{TRAIL}}$ coordinate the initialization, enabling and latching of the 16 bit binary up counters, which characterize the DLR signal. The 74F269 eight bit counters are cascaded to obtain the 16 bit binary up count for either the low interval period (IC_13A, IC_7B) or the high interval period (IC_9A, IC_11A) of the DLR signal. These 16 bit up counters are latched into the 74F374 octal latches (i.e., IC_10A, IC_12A high interval count and IC_8B, IC-14A low interval count). The FAST series logic is selected for speed, high drive, and low noise characteristics.[4]

The timing diagram in Figure 3 shows the sequence of events that occur during the time interval measurement with the TIC circuit. The high-to-low transition of the $\overline{\text{LEAD}}$ signal occurs on the first positive oscillator clock transition after the DLR signal becomes an active high. The high-to-low transition on $\overline{\text{LEAD}}$ parallel loads zeros into the high interval counters IC_9A and IC_11A. The low-to-high transition on $\overline{\text{LEAD}}$ enables the high interval counter for counting and latches the value of the low interval counters. High interval counting continues until the high-to-low transition of the $\overline{\text{TRAIL}}$ signal. The high-to-low transition on the $\overline{\text{TRAIL}}$ signal parallel loads zeros into the low interval counters IC_13A and IC_7B. The low-to-high transition on the $\overline{\text{TRAIL}}$ signal enables the low interval counters for counting and latches the value from the high interval counters. The asynchronous relation between the DLR signal and the time oscillator time clock base permits an interval count error, which is inherently less than two oscillator clock periods. Gating the counter clock inputs with the terminal count from the most significant 8 bit counters prevents erroneous counts due to counter roll over. Recording each interval count without loss requires an ICBM circuit.

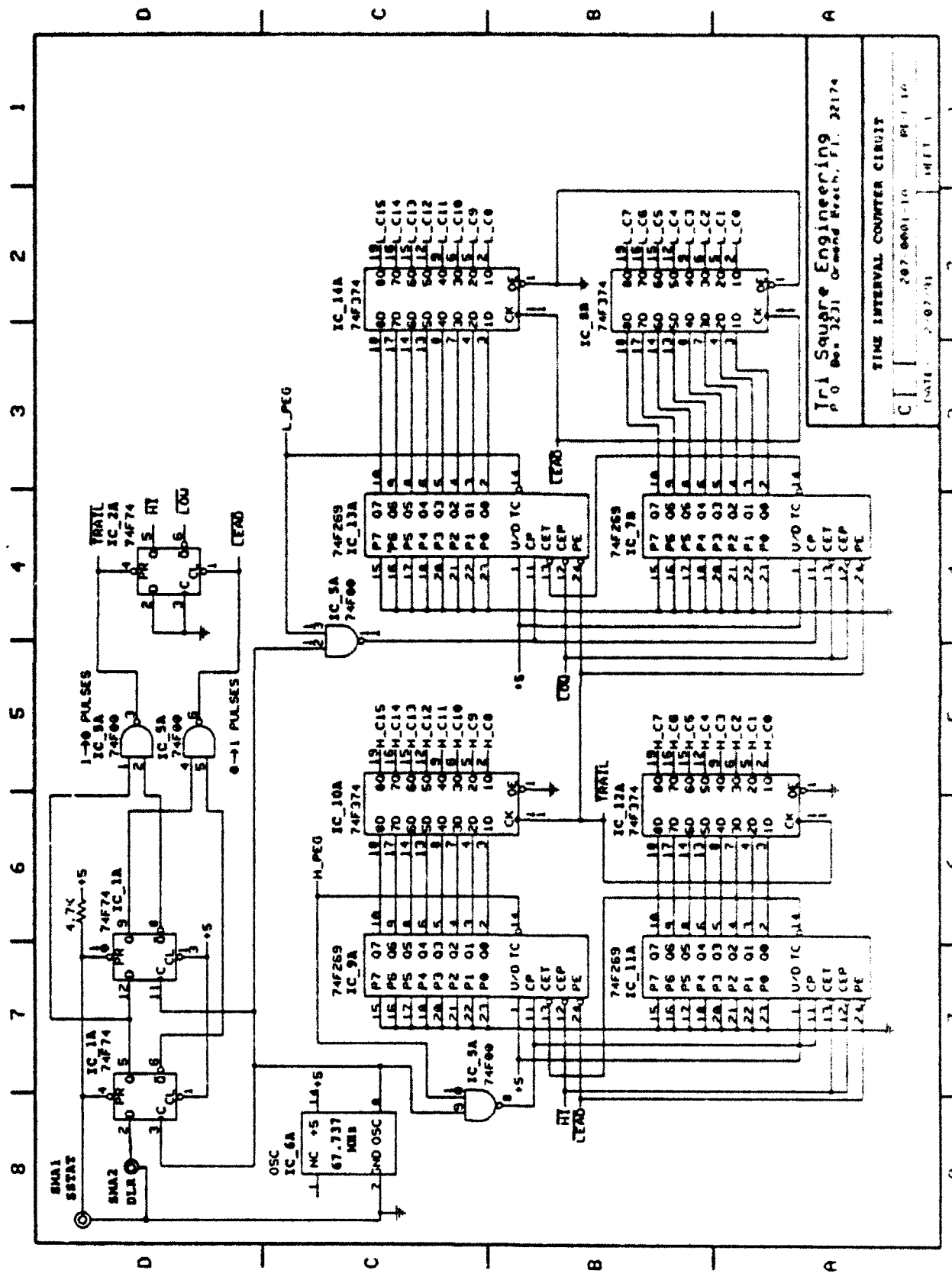

Figure 2. Electronic Wiring Diagram of the TIC Circuit

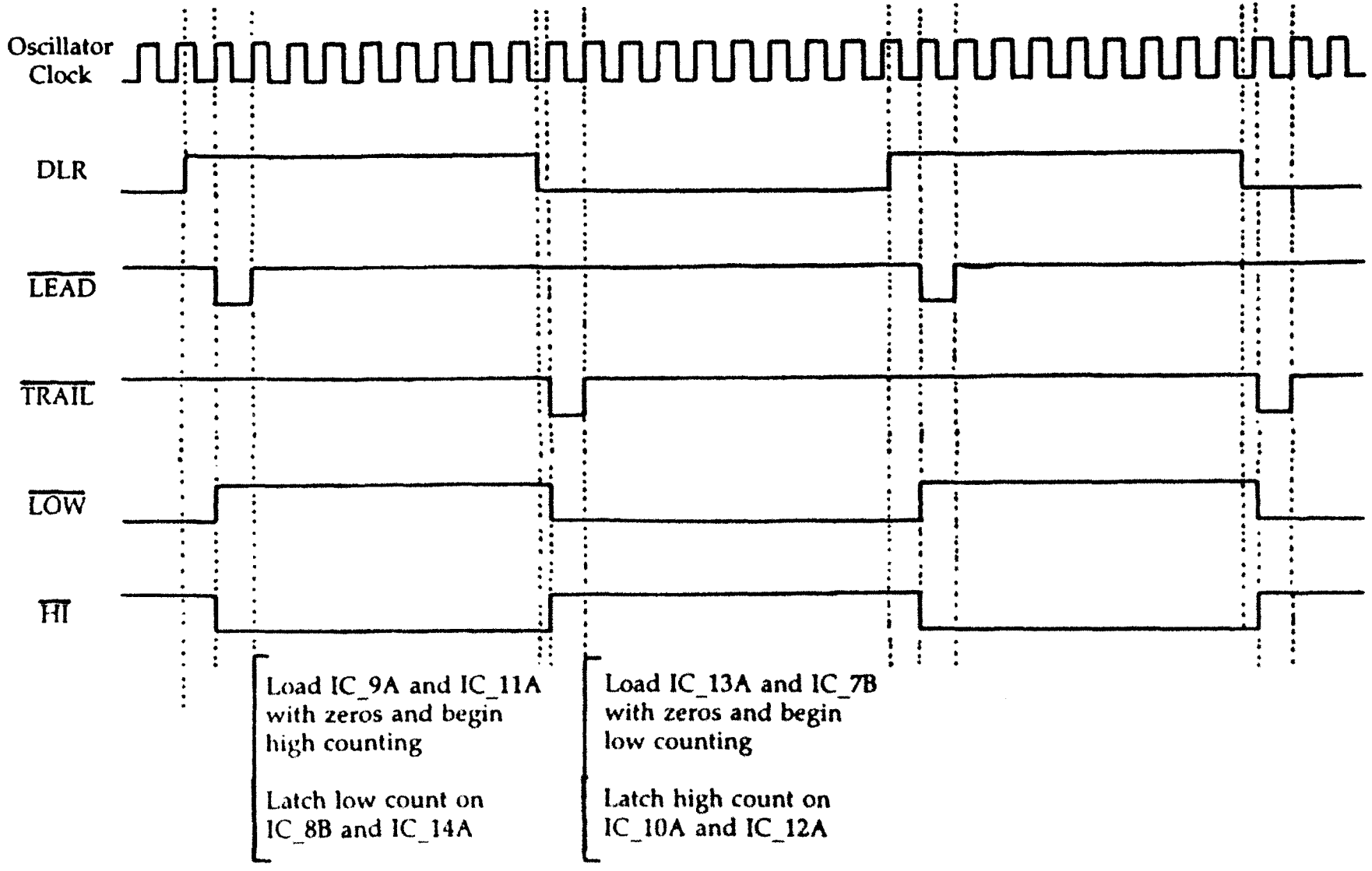

Figure 3. Timing Diagram Showing Relationship Between the TIC Circuit Control Signals

2.2 ICBM Circuit.

The ICBM circuit performs two functions. First, the ICBM circuit records up to 4096 consecutive interval counts for both the high and low time intervals. Second, an interface to the IBM/PC-XT bus permits loading the tic into the computer. This design eliminates the possible loss of tic and permits access to interval counts upon occurrence.

The ICBM circuit schematic is shown in Figure 4. The four IDT7204 First-In/First-Out (FIFO) memory integrated circuits (IC) provide a memory buffer for both high and low interval counts. Each FIFO memory IC is configured as 4096 nine bit words. The high and low interval counter latches are connected to the FIFO data inputs. FIFO IC_5B and IC_3B are arranged to contain the high interval counts, while FIFO IC_6B and IC_4B hold the low interval counts. The TIC circuit control signals of $\overline{LOW}$ and $\overline{HI}$ write the respective low and high interval counts from the octal latches in FIFO memory. Each FIFO requires pin 7, the expansion in ($\overline{XI}$), to be grounded and pin 23, the first load/retransmit ($\overline{FL}/\overline{RT}$) input to be tied at +5 V. A ground on the $\overline{XI}$ input places the FIFO in the single device operation mode. A low pulse on the $\overline{FL}/\overline{RT}$ input sets the internal FIFO read pointer to the first memory location but does not affect the FIFO write pointer. Therefore, a connection of +5 V to the $\overline{FL}/\overline{RT}$ input avoids a request to retransmit the FIFO contents when in the single device operation mode. The ICBM circuit provides the FIFO full or empty status outputs to the IBM/PC through an octal buffer IC_8A. An active FIFO full flag ($\overline{FF}$) output inhibits further write operations. The $\overline{FF}$ can also indicate the absence of data reads by the IBM/PC. An active FIFO empty flag ($\overline{EF}$) output occurs after the FIFO contents are completely read or a reset is performed.

Input/Output (I/O) addressing by the IBM/PC permits reset, status check, and data content read operations for each FIFO. The programmable array logic (PAL) device, IC_7A (i.e., PAL16L8) decodes the lower 4 address bits during the appropriate device select signal. The PAL assigns individual I/O addresses to each I/O device. The device select signal to the PAL is provided by the JDR PR-2 (JDR Microdevices; Los Gatos, CA) IBM/PC-XT bus prototype card on which the TIC and ICBM circuits are constructed. The prototype card generates eight device select signals from the lower 10 address bus bits and the I/O bus control signals. The base address of 300 hexadecimal is selected since it is reserved for prototype card design development. The card provides a bidirectional connection to the data bus through an octal bus transceiver (i.e., 74LS245). Loading the interval counts from FIFO memory into the IBM/PC via the data bus requires the program described in the next section.

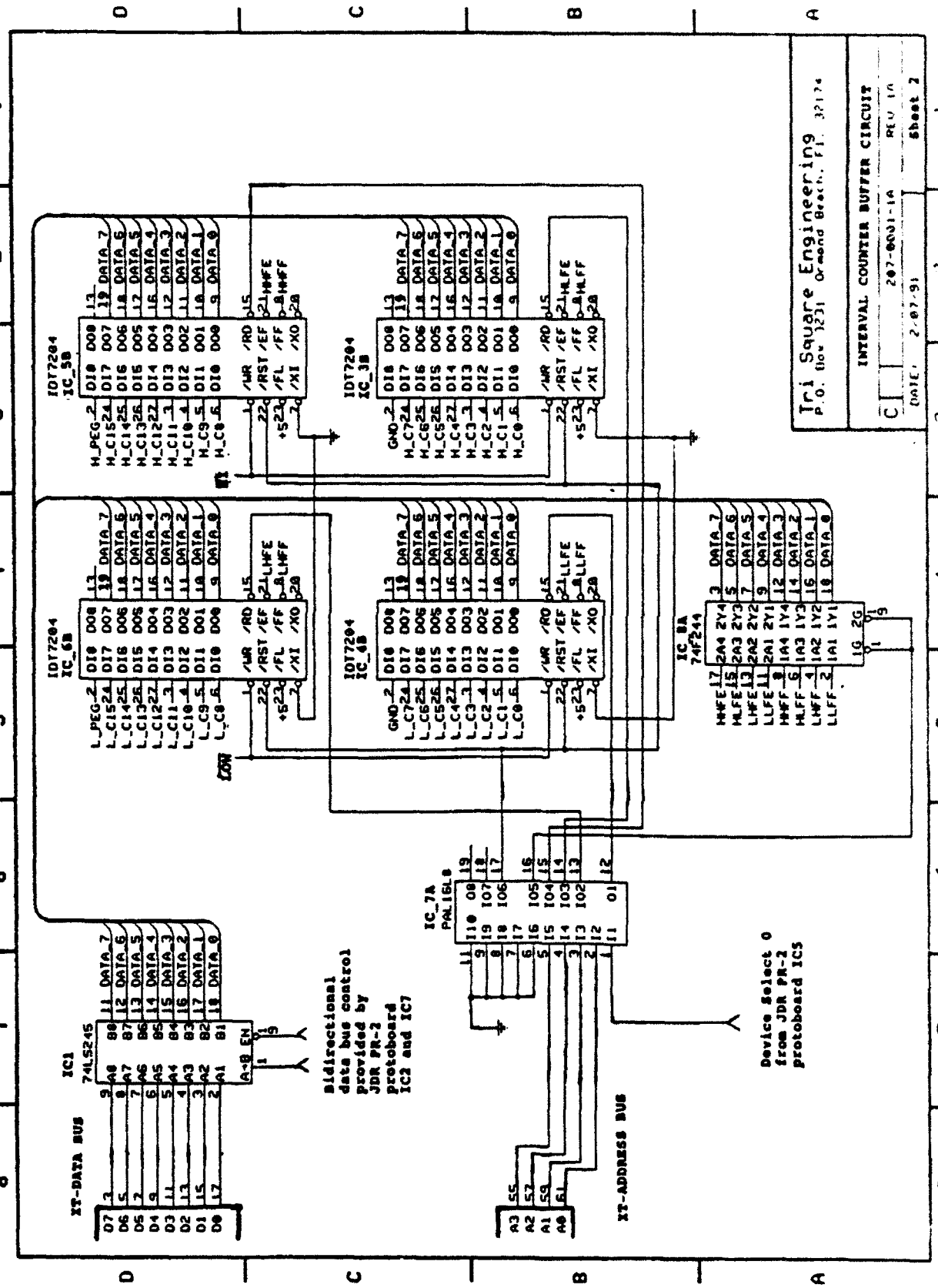

Figure 4. Electronic Wiring Diagram of the ICBM Circuit

2.3 TICOUNT Program.

The TICOUNT program performs three tasks. First, the program insures the proper collection of the tic. Second, the program transfers data from the ICBM circuit to the IBM/PC. Third, the program performs a statistical analysis of the tic collected. These tasks are outlined in Figures 5a and 5b.

The TICOUNT program avoids improper collection of the time interval counts by monitoring a status register containing the SSTAT signal. Input port 306 hexadecimal identifies the status register that contains the SSTAT signal. The interferometer generates the SSTAT signal for each mirror scan. An active SSTAT signal (i.e., logic 1) indicates when the DLR signal is valid. Monitoring the SSTAT signal permits determination of the beginning of each interferometer mirror scan. The FIFO memories are reset until a valid SSTAT signal is received. Addressing I/O port 304 hexadecimal resets the FIFO memory. This sets the internal FIFO read/write pointers to the first memory location. Time interval count collection begins after a transition of the SSTAT signal from low to high. A total of 8192 sixteen bit interval counts can be written into the FIFO memory with the present configuration.

Once the FIFO memory is loaded with the first tic, the program begins reading the counts into the IBM/PC. The consecutive list of input port hexadecimal values from 300 to 303 are assigned to the FIFO memory. The low and high bytes of the 16 bit low interval counts are located at input ports 300 and 301, respectively. The low and high bytes of the 16 bit low interval counts reside at input ports 302 and 303, respectively. Each interval count must be read before a time-out condition occurs. Occurrence of the time-out during the C program execution in program listing 1 (Appendix A) results in an error message. If a time-out does not occur, then the FIFO memory is read until the FIFO register indicates empty. The FIFO status register indicates the empty and full flags condition at input port 305 hexadecimal. The FIFO status register holds a hexadecimal value of 0F after all interval counts have been read.

After the time interval counts for a mirror scan are loaded into the IBM/PC memory, the third TICOUNT program task begins. Analysis of the tic collected can be performed. The two finite sample size statistics of average and best estimate of the standard error serve to represent the collection of time interval counts over one interferometer mirror scan. Two implicit assumptions are made in calculating the tic average and its variation. First, the error associated with the method of time interval measurement is negligible compared to the inherent variation in the time intervals. Second, a Gaussian distribution of the tic is present. The average tic represents the best estimate of the mean mirror velocity over one complete mirror

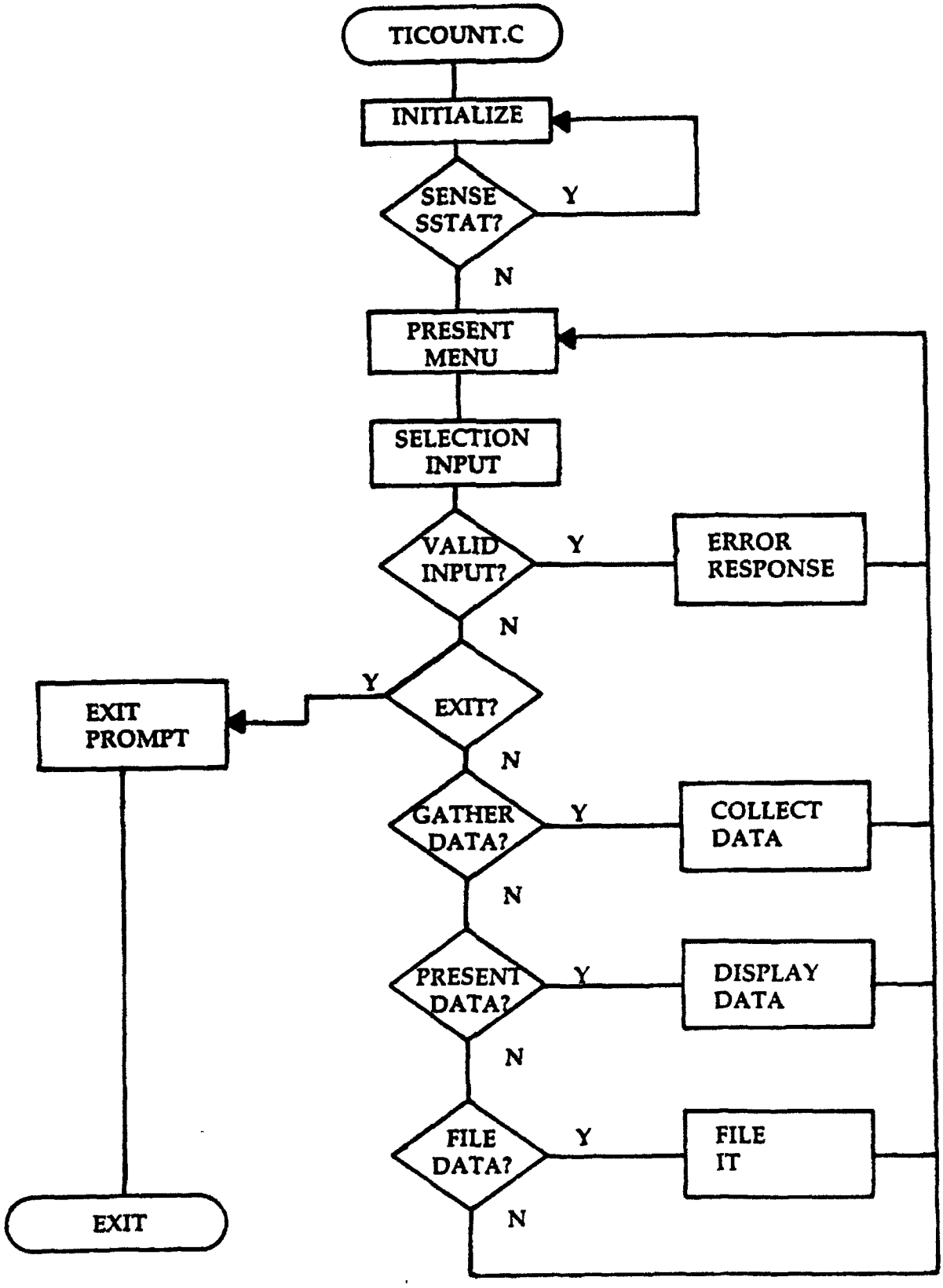

Figure 5a. The Program TICOUNT.C Reads Each Time Interval Count (tic) Very Rapidly from the ICBM Circuit. TICOUNT.C also Permits storage of the tic values on disk

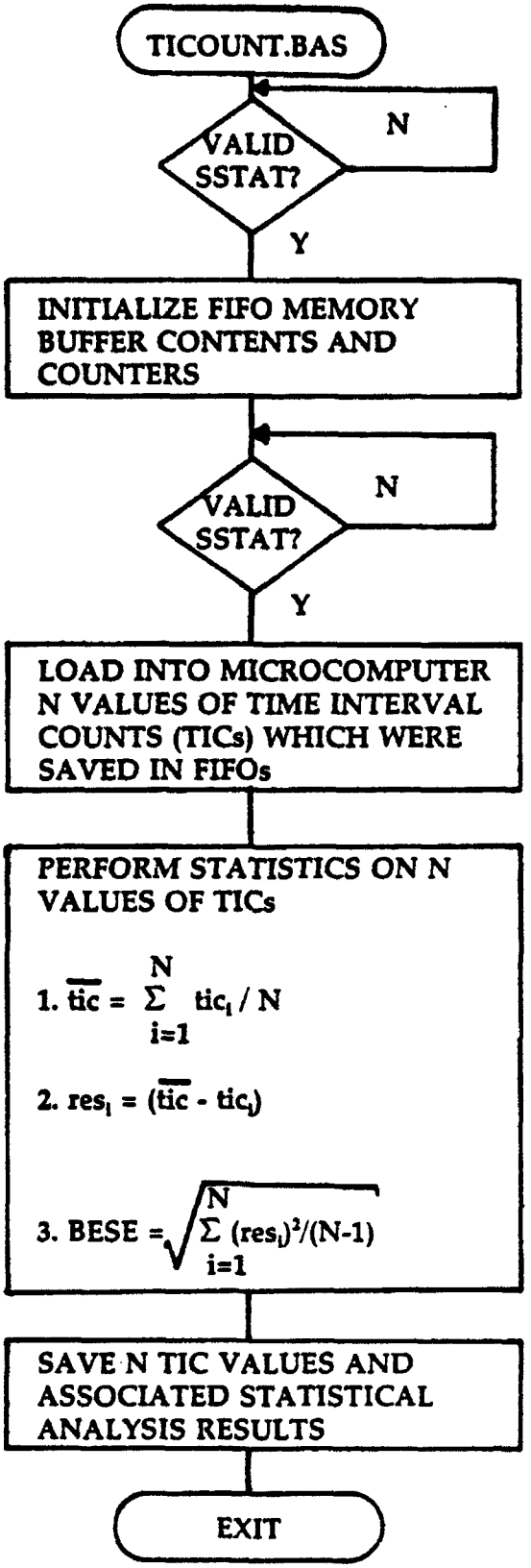

Figure 5b. The Program TICOUNT.BAS Reads Each tic from the ICBM Circuit and Performs an Analysis on the tic Values

scan. The difference of the individual tic from the average count (i.e., residual count value) permits calculation of mirror velocity variation over the mirror scan length. The residual count values are used to calculate the best estimate of the standard error (BESE). The TICOUNT flow chart in Figure 5 outlines the procedure for calculation of the BESE. Program Listing 2 (Appendix B) lines 5000 to 5640, perform the statistical analysis in Basic. Upon completion of the statistical analysis, the TICOUNT program allows the user to save or print the tic and associated analysis.

3. CIRCUIT ASSEMBLY

The TIC and ICBM circuits are constructed on the perforated wire wrap portion of the prototype card (JDR Microdevices). The address, data, and control signal buffering/decoding are provided on the printed circuit board (PCB) portion of the prototype card (i.e., JDR PR-2 card).[5] The JDR PR-2 card contains a power/ground distribution grid on the wire wrap portion of the card for electromagnetic interference (EMI) suppression. The decoupling capacitors C1 through C10 are soldered to the power/ground distribution grid with Vector T44 wire wrap pins. Integrated circuit power supply connections are made with wire wrap to the decoupling capacitor wire wrap ports.[6] Component placement on the JDR PR-2 board is shown in Figures 6a and 6b. The PCB portion of the JDR PR-2 board is located immediately above the bus edge connector. This portion of the JDR PR-2 board is populated with integrated circuits IC1 through IC7. The resistors, RP1-RP4, and capacitors, CP1-CP7, are also located on the PCB portion of the prototype board along with a 4 position SPST switch, SW. Any integrated circuit designation with a letter suffix refers to the TIC or ICBM circuitry. The subminiature series A (SMA) coaxial connectors, in the upper right hand corner of Figure 6a offer a compact input of the SSTAT and DLR interferometric signals to the prototype card. Table 1 lists and describes the components necessary for the TIA circuit construction.

4. TIME INTERVAL MEASUREMENTS

The time interval measurements that are made with the TIA permit the incremental determination of the interferometer mirror velocity. Correct operation of the TIA circuitry is tested with the input of known time intervals. These time intervals are synthesized with a signal simulator. Once the TIA operation is validated with the signal simulator, the TIA circuitry is connected to the interferometric SSTAT and DLR signals. The tic are collected for the DLR signal over one

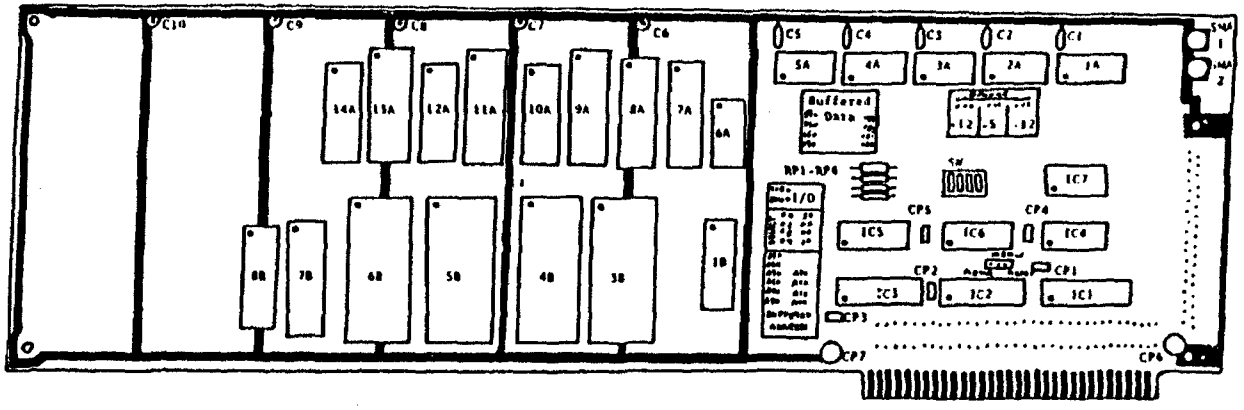

Figure 6a. Top View from the Component Side of the Prototype Circuit Board, which Shows Component Placement on the IBM-PC Compatible Circuit Card

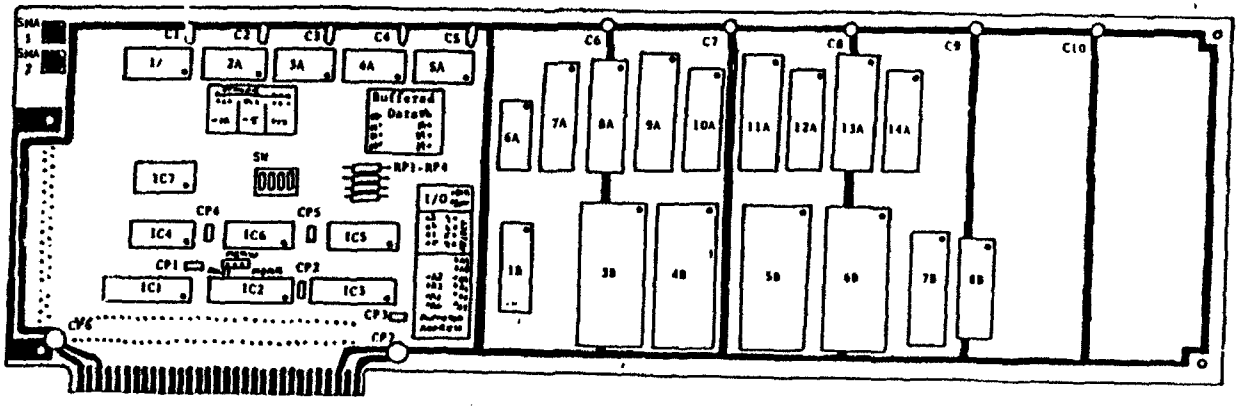

Figure 6b. Bottom View from Circuit (i.e., Wire Wrap) Side of the Prototype Circuit Board, which Provides a Reference for Circuit Construction and Testing of the IBM-PC Compatible Circuit Card

Table 1. Circuit Component Values and Descriptions

IC #	Description	Power Supply Pin Connections +5 V	GND
1A,2A	74F74, Dual Data Flip Flop	14	7
3A, 7	74LS08, Quad AND Gates	14	7
4A	74F04, Hex Inverters	14	7
5A	74F00, Quad NAND Gates	14	7
6A	F3000, FOX 67.73760 MHz TTL Oscillator	14	7
7A	PAL16L8, combinatorial programmable array logic device	20	10
8A,1B	74LS244, octal buffer	20	10
2, 3	74LS244, octal buffer	20	10
9A,11A	74F269, 8 bit counter	19	7
13A,7B	74F269, 8 bit counter	19	7
10A,12A	74F374, octal latch/buffer	20	10
14A,8B	74F374, octal latch/buffer	20	10
3B,4B	IDT7204L, FIFO Memory 4096X9 bit	28	14
5B,6B	IDT7204L, FIFO Memory 4096X9 bit	28	14
1	74LS245, octal bus transceiver	20	10
4	74LS00, Quad NAND gates	14	7
5	74LS138, 3 to 8 Line Decoder	16	8
6	74LS08, Quad AND gates	14	7

Capacitor	Capacitance Value in µf	Resistor	Resistance Value in Kohm
C1-C10	0.01	RP1-RP4	4.7
CP1-CP5	0.01		
CP6&CP7	10.00		

Other Components

SW	4 position SPST switch
SMA1, SMA2	right angle PCB coaxial receptacle

mirror scan. These counts are used to calculate the interferometer mirror velocity. Comparison of the tic that are collected for sequential mirror scans shows any repeatable mirror velocity behavior.

Accurate TIA operation is demonstrated with an interferometer signal simulator. A polynomial waveform synthesizer (PWS) and monostable circuit permit simulation of the interferometric SSTAT and DLR signals. These simulated signals verify that the TIA is operating properly. A block diagram of the interferometer simulator is shown in Figure 7. A monostable circuit with a time constant of approximately 100 ms generates the valid data flag (i.e., SSTAT signal), which triggers the PWS. The +5 V and 10 KHz^2 wave configuration for the PWS (Figure 7 selected settings) produces a gated DLR signal similar to the interferometer sampling signal.[7] For a +5 V and 10 KHz^2 wave, the tic is calculated from the ratio of 100 μs/14.76285 ns and is found to be 3386.89 counts. The average tic of 3386.96 is obtained from the TIA with a BESE of ~ 0.40. A count of 3387 is most probable since only an integer number of counts are recorded. A plot of the individual tic variation from the average interval count of 3386.96 is shown in Figure 8a. The largest residuals occur in the first 100 tic (Figure 8b). An overall residual variation of 6 counts is still within the error of the 100A ns clock increment generated by the PWS (e.g., 100 ns/14.76285 ns = 6.8 counts). The distribution of the tic that are obtained from the PWS, is shown in Figure 9. The interval counts responsible for the 6-count variation in the PWS data constitutes only 1% of the interval count data. The residuals within 1 count from the average interval count comprise about 9% of the interval count population. The remaining ~ 90% of the tic population occur at an interval count of 3387. The TIA gives an average interval count for the simulated DLR signal that is with 0.002% of the predicted interval count. The predicted count also falls within the ±0.01% variation that is calculated from the BESE of the tic population. Clearly, the analysis of the 10 KHz simulated DLR signal indicates that the TIA performance is adequate for measuring the time intervals associated with an interferometer sampling at 10 KHz.

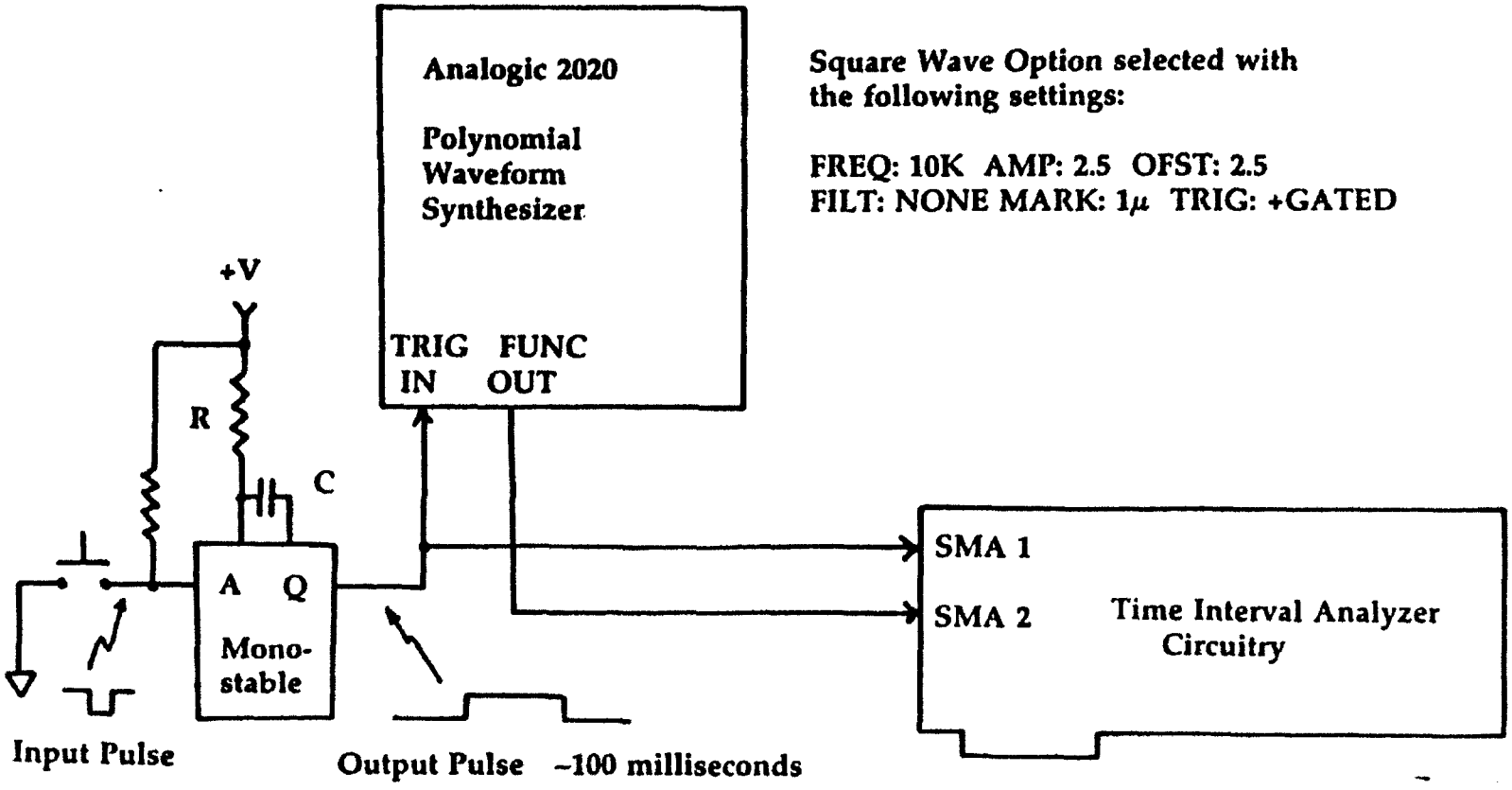

Figure 7. Interferometer Signal Simulation with a Polynomial Waveform Synthesizer and Monostable Circuit Permits Verification that the Time Interval Analyzer is Operating Correctly

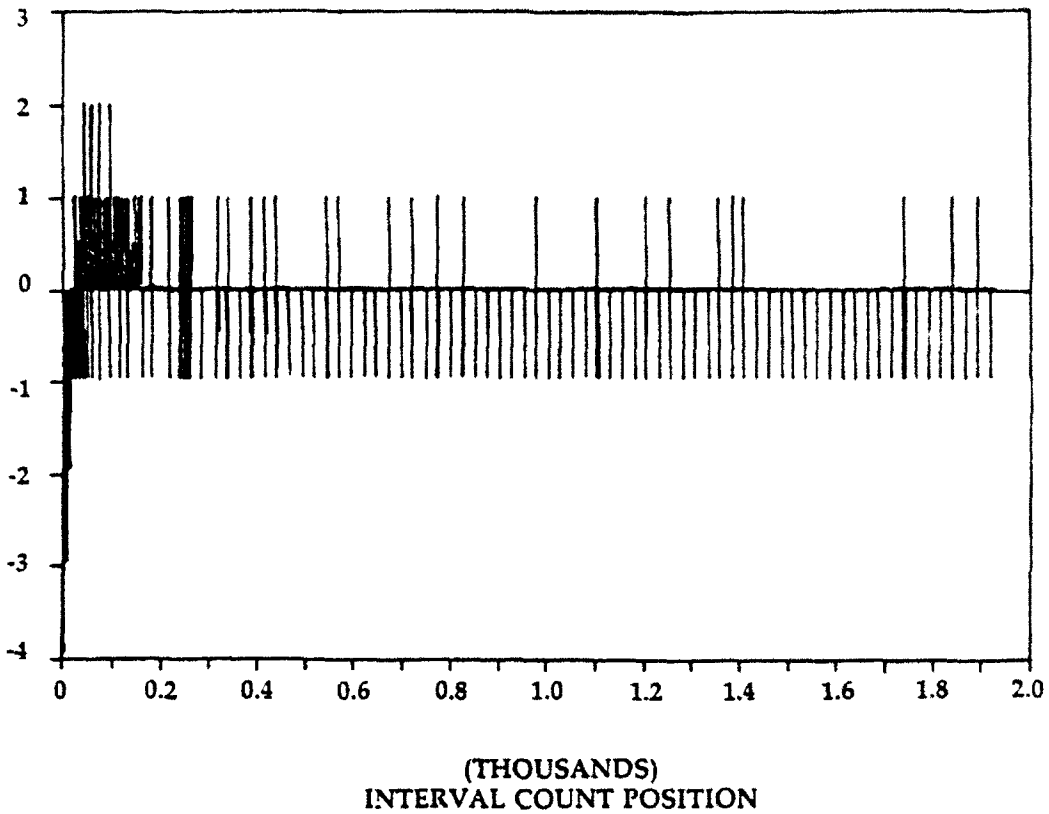

Figure 8a. The tic Variation for a 10-kHz Simulated Digital Laser Reference Signal with ~100 ms Duration

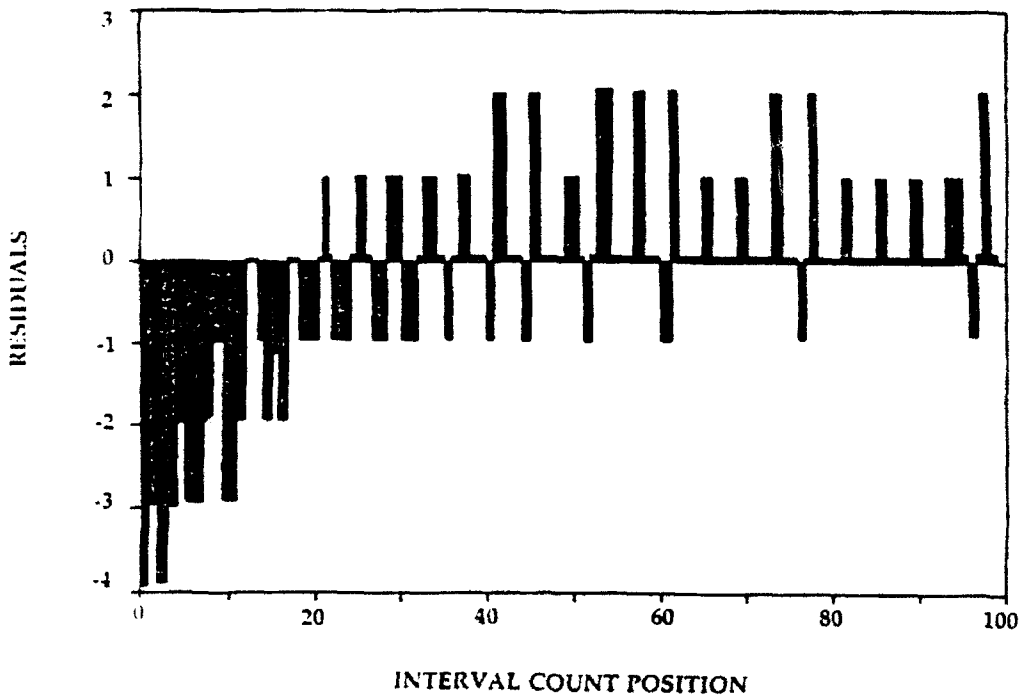

Figure 8b. The Largest Variation in the Simulated Digital Laser Reference Signal Occurs During the First 0.5 ms

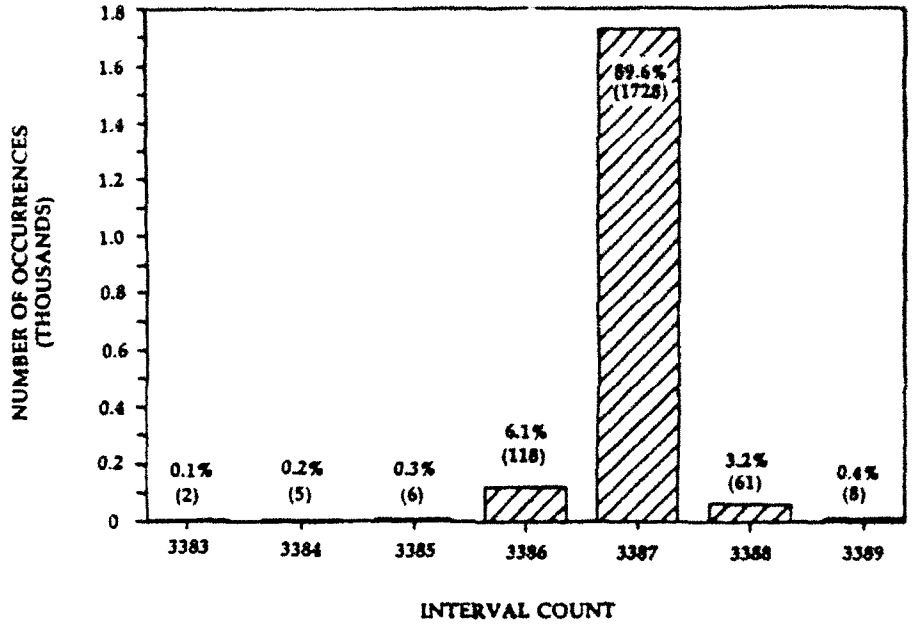

Figure 9. Distribution of tic from the Interferometer Signal Simulator. The average and ±BESE are 3389.96 and ±0.40, respectively.

The tic from the interferometric DLR signal are summarized in Figures 10 and 11. The average tic of a single interferometer mirror scan is 3354 with a BESE of ±19. The variation from the average tic is shown in Figure 10a as a plot of the residuals versus the tic position. This residual plot shows a sinusoidal behavior, which is expected for a servo mirror controller attempting to maintain a constant mirror velocity over the mirror scan. Figure 10b is an expanded view of the residuals from the interval count positions 200 to 350. The largest positive residual occurs at an interval count location of 254. The relationship between the tic and mirror velocity is given by equation 1.

$$u\ (cm/s) = \left[SI\ (cm)/\left[\overline{tic} * toc\ (s)\right]\right]/2 \qquad (1)$$

where

u = mirror velocity, cm/s

SI = sampling interval, cm
[2*632.8(10^{-9}) m*100 cm/m]

tic = average time interval count; counts

toc = time of oscillator clock, s
[67.7576(10^{6})H→14.76285 ns]

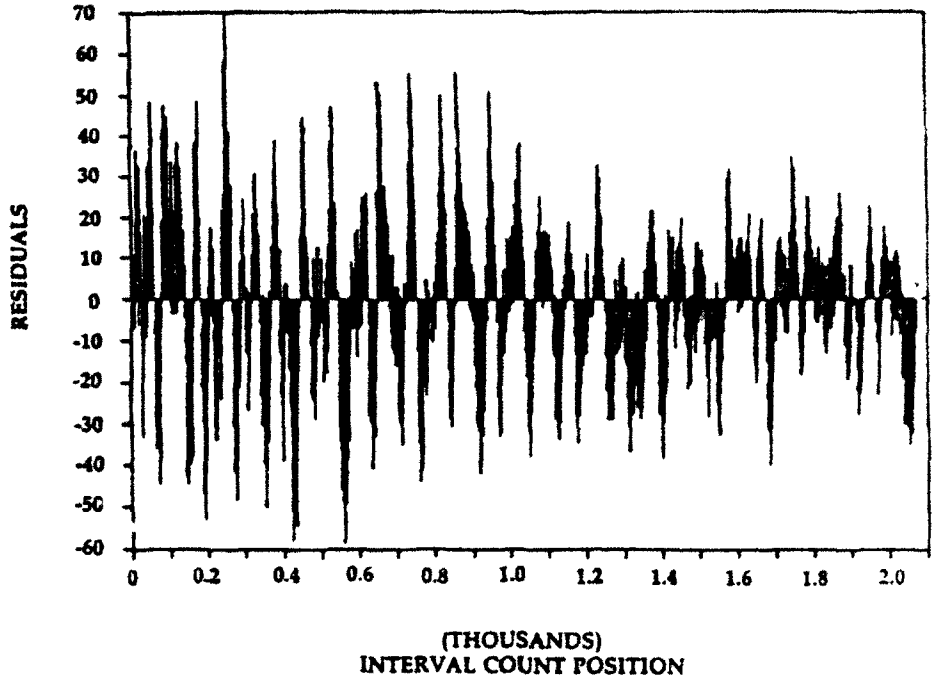

Figure 10a. Variation of the Interferometric DLR Signal Plotted as a Function of Mirror Displacement in the Michelson Interferometer. The entire mirror translation is shown.

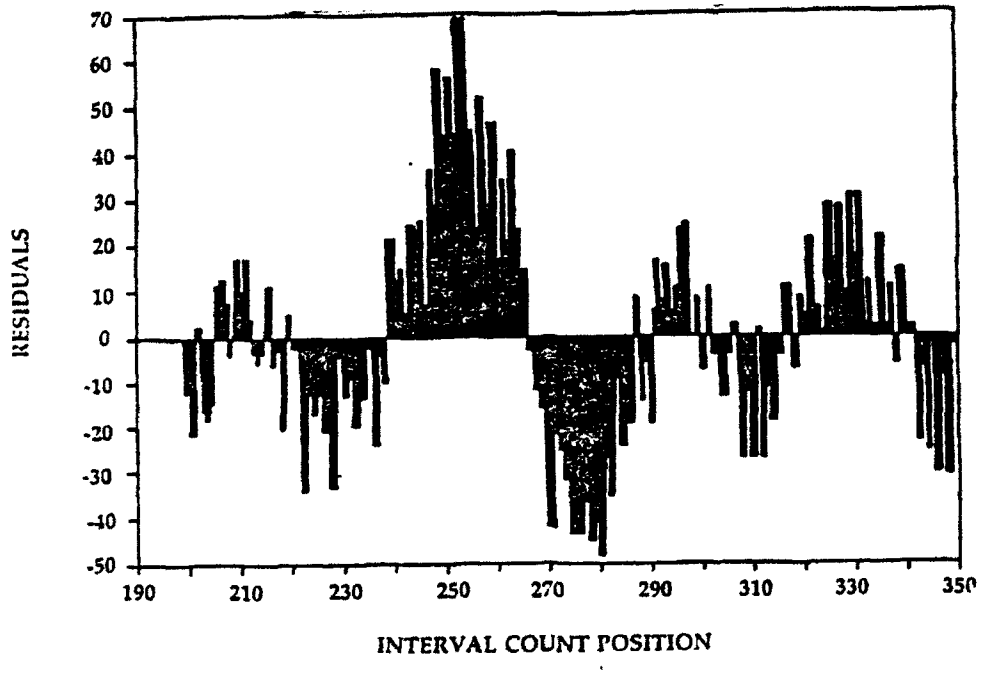

Figure 10b. Largest Variation in the DLR Signal from the Interferometer Occurs at Approximately Interval Count Position 260.

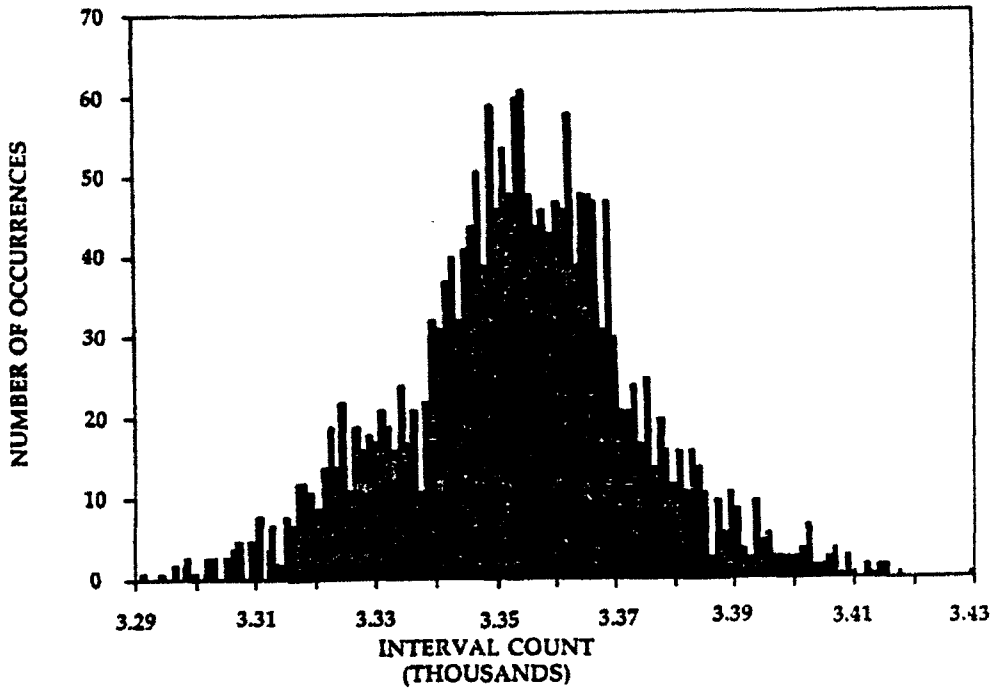

Figure 11. Distribution of Interval Counts Obtained from the Interferometric DLR Signal. The average count of 3354 is obtained with a ±BESE of ±19.

The sampling interval is based on the HeNe laser wavelength of $1.265(10^{-4})$cm. A level transition in the DLR signal occurs on every second positive zero crossing of the laser fringe. Therefore, a factor of two must be introduced for a correct sampling interval. The average tic is 3354 for the interferometer scan shown in Figures 10 and 11. The time base of the oscillator clock is 14.76285 ns. The factor of two in the denominator of equation 1 occurs since the optical retardation of the interferometer is twice the mirror velocity.[2] The distribution of the tic is shown in Figure 11. Approximately two-thirds of the tic fall within ±19 counts of the average tic, 3354. Calculation of mirror velocity with equation 1 for eight sequential interferometer mirror scans is tabulated in Table 2. A mirror velocity of 1.277 cm/s with a variation of ±0.5% is found. This velocity is consistent with the reported specification value of 1.25 cm/s.[8]

Table 2. Interferometer Mirror Velocity

$\overline{tic}$ ± BESE (counts)	u ± BESE (cm/s)
3354 ± 19*	1.277 ± 0.007
3354 ± 18	1.277 ± 0.006
3354 ± 17	1.277 ± 0.006
3355 ± 17	1.277 ± 0.006
3355 ± 16	1.277 ± 0.006
3355 ± 18	1.277 ± 0.007
3355 ± 16	1.277 ± 0.006
3355 ± 16	1.277 ± 0.006

*tic plotted in Figures 10 and 11.

The eight sequential mirror scans permit the determination of any repeatable behavior in the incremental mirror velocity. Figure 12a provides a plot of the average interval count ±BESE as a function of interval count position for interval count positions 200-300. The variation of mirror velocity across the interval count positions is larger than the variation at each count position. This tends to support the hypothesis that the interferometer mirror is controlled in the same manner from scan to scan. This repeated behavior may be due in part to the use of a white light source detection scheme to indicate the first valid mirror position. The white light source signal generates the SSTAT signal at the same mirror location on initiation of each mirror scan. The average of 8 interval counts is represented by a blank space. This blank space is bracketed by vertical bars representing ±BESE. Figure 12b is a plot of the average of 8 interval counts ±BESE for the entire mirror scan. The blank spaces of the average interval count are connected. These blanks sweep out an area from the plot of the average interval count ±BESE. The individual interval count variation from scan to scan is approximately ±15. However, the overall interval count variation in the first half of the mirror scan is about twice as large as the second half of the mirror scan. This overall variation in the first half of the mirror scan is approximately ±65 counts, which is approximately four times larger than any variation at any particular interval count position.

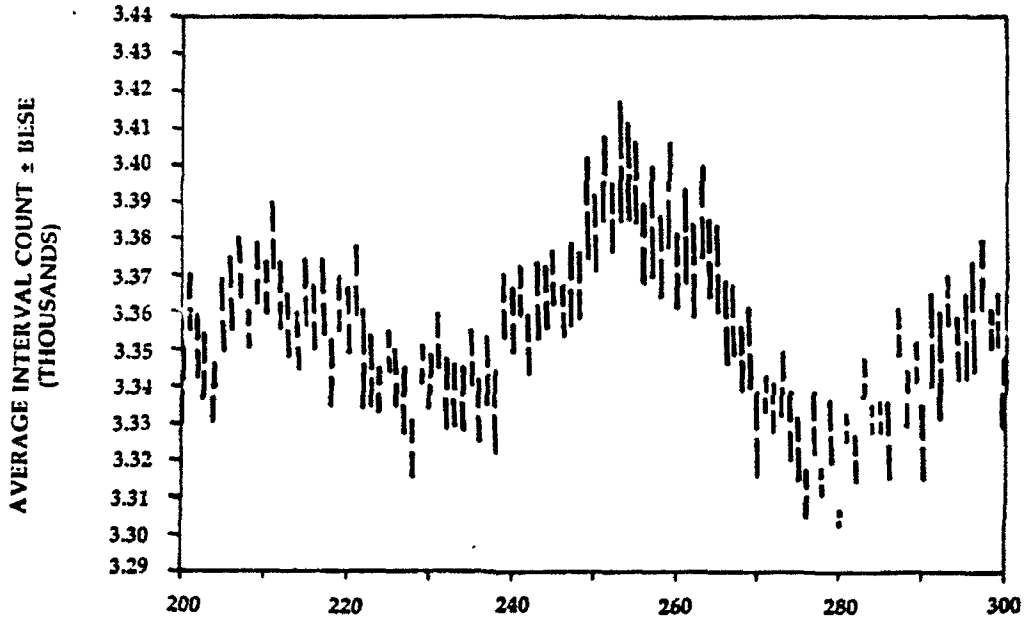

Figure 12a. Scan-to-Scan Repeatability is Shown. For eight sequential mirror scans, the average interval count and ±BESE (i.e., blank space bracketed by two vertical lines) at a particular interval count position are plotted.

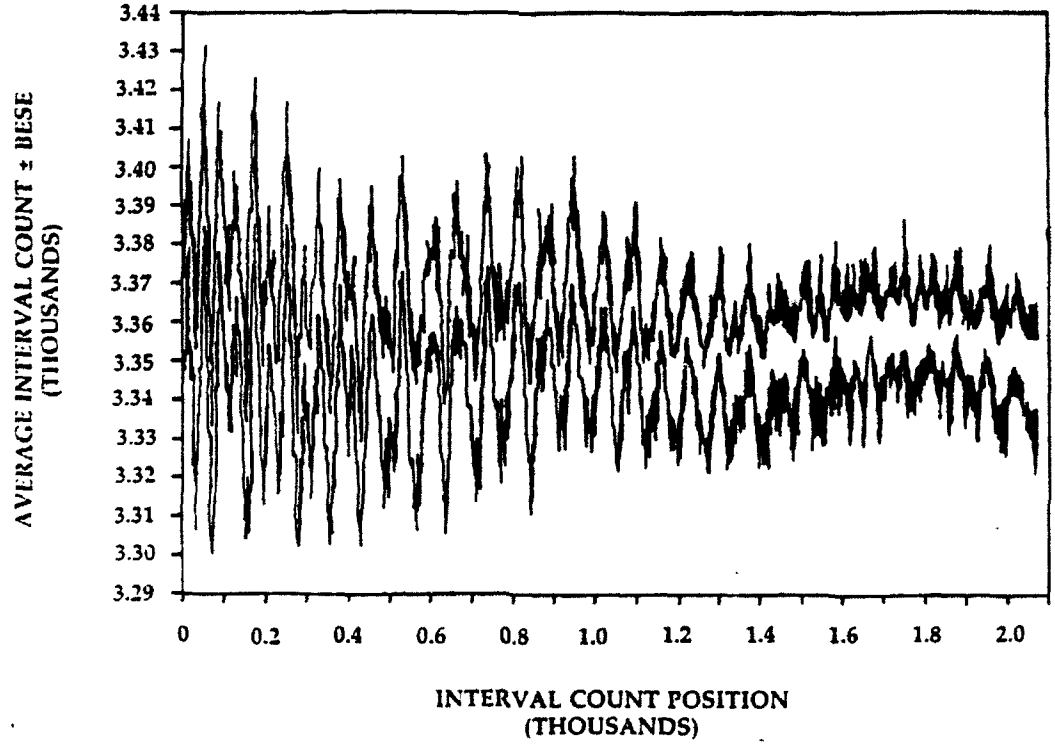

Figure 12b. Scan-to-Scan Repeatability over the Entire Mirror Displacement Shows Twice as much Variation in the First Half of the Mirror Scan as Opposed to the Second Half.

5. CONCLUSIONS

Sufficient information has been supplied to construct a time interval analyzer (TIA) and to perform the initial TIA testing with a signal generator. Documentation is provided on all circuitry and programs that are implemented in the TIA. Selection of the IBM/PC-XT circuit card format can facilitate integration of the TIA into a Fourier transform infrared (FTIR) spectrometer, which operates in a high vibration environment. The TIA has measured an interferometer mirror velocity of 1.277 cm/s, which compares well with the reported value of 1.25 cm/s. The $\pm$0.5% variation in the mirror velocity is less than the $\pm$2% variation cited in reference 2. The $\pm$2% variation can be responsible for a significant decrease in the interferogram signal-to-noise ratio.

There are two potentially useful improvements to the TIA. First, a more accurate measure of time intervals is possible by use of interpolation. Interpolation can reduce by a factor of 200 (i.e., tens of picosec time resolution) the maximum inherent count uncertainty value of two oscillator clock cycles. The count uncertainty occurs due to the asynchronous relation between the DLR signal and the oscillator clock time base. The necessary circuitry for interpolation consists of a constant current source, a small capacitor, a sample/hold circuit, and an analog-to-digital converter. The capacitor voltage that is read with an analog-to-digital converter is directly proportional to some fractional oscillator clock cycle.[3] A second consideration is the ability to input the helium-neon (HeNe) laser reference signal in the analog form. The ability to handle the analog signal is desirable, since many commercial FTIR spectrometers provide laser indexing in the analog format. This requires impedance matching the input signal to the appropriate voltage comparator. The voltage comparator must possess sufficient speed to provide an oscillation free representation of the HeNe sinusoidal zero crossings.

LITERATURE CITED

1. Masters, L.W., "Use a Counter to Characterize Rapidly Changing Signals," EDN Vol. 35 (4), pp 159-166 (1990).

2. Griffiths, P.R., and deHaseth, J.A., "Fourier Transform Infrared Spectroscopy," In Chemical Analysis, Vol. 83, pp 262-267, P.J. Elving and J.D. Winefordner, Eds., John Wiley and Sons, New York, NY, 1986.

3. Willison, J., "Counters/Timers: Precision Frequency and Time-Interval Measurement," Evaluation Engineering Vol. 30 (6), pp 42-51 (1991).

4. Hall, W., "Avoid Confusion in Choosing Digital Logic," Electronic Design Vol. 39 (20), pp 63-76 (1991).

5. JDR PR-1 and PR-2 Users Manual, JDR Microdevices, Los Gatos, CA, 1986.

6. Moore, J.H., Davis, C.C., and Coplan, M., Building Scientific Apparatus, pp 455-457, Addison-Wesley Publishing Company, London, England, 1983.

7. User's Manual for Polynomial Ad Synthesizers Models 2020 and 2000, 82-5016 Rev C/Software Rev 4.01, Analogic Corporation, Data Precision Division, Peabody, MA, 1990.

8. Bryson, R., "XM21 Remote Sensing Agent Alarm," Workshop on Remote Sensing for Chemical Defense, DAAG-29-81-0-0100, Myrtle Beach, SC, April 1985.

Blank

APPENDIX A
PROGRAM LISTING 1. TICOUNT.C

```
/*************************************/
/* TICOUNT Program                   */
/* Collects time interval counts     */
/* Provides Display of data          */
/*-----------------------------------*/

#include  <gatebrd.h>
#include  <stdio.h>
#include  <dos.h>

/* GLOBALS */
FILE *hi_lo;
unsigned int period_hi[1024]; /* PERIOD HIGH SAMPLES */
unsigned int period_lo[1024]; /* PERIOD LOW SAMPLES */
unsigned int data_taken;      /* DATA TAKEN QUALIFIER */
unsigned int max_samnum_lo;   /* MAXIMUM SAMPLE NUMBER */
unsigned int max_samnum_hi;   /* MAXIMUM SAMPLE NUMBER */

/*************************************/
/*MAIN()                             */
/*************************************/

main()
  {/* MAIN LOOP */
  int  iterate;             /* ITERATION FOR TEST*/
  int  perform;             /* PERFORMANCE INDICATOR*/
  int  option_choice;       /* FUNCTION CARRIER */

  iterate = 1;              /* SET FOR FAILURE & ITERATION*/

  while(iterate)            /* CONTINUE TILL INITIATE IS CORRECT */
    {
    perform = init();       /* INITIATE & CHECK FOR CORRECT*/

    if(perform == TRUE)     /* INITIATE IS CORRECT */
      {
      iterate = FALSE;      /*KILL ITERATION*/
      printf("BOARD INIT SUCCESS!\n");
      }
    else
      {
      printf("Gate board INIT FAILURE!\n");
      }
    }

  printf("\nTomCat's GATE BOARD TEST UTILITY v1.0\n");
  option_choice = 'L';      /*SET FOR COMMAND PERFORM*/

  /*SELECT AN OPTION & MAKE UPPER CASE*/

  while(option_choice != 'X') /*GO TILL X SELECTED*/
    {
    option_choice = toupper( pick_it_out());

    /* EXECUTE OPTION AND INDICATE OUTCOME*/
    perform = execute(option_choice);

    if(perform == TRUE)     /* THE OPTION SELECTED WAS PERFORMED*/
      {
      printf("OK!\n");      /* INDICATE SUCCESS */
      }
```

```
    }
  printf("Program terminated!\n");
  }

/************************************/
/*INIT()                            */
/************************************/
int init()
  (/*INITIALIZATION ROUTINE*/
  int perform;              /*PERFORMANCE INDICATOR*/
  int stat;                 /*COMPARISON HOLDER*/

  stat = inp(STATUS)& HHFF;/* GET THE OVERFLOW STATUS */

  if(stat == FALSE)
    {                       /* AN FULL FLAG (0) INDICATES SUCCESSFUL INPUTS */
    perform = TRUE;         /* INDICATE SUCCESS */
    data_taken = FALSE;     /* PRESET TO INDICATE NO DATA */
    }
  else
    {                       /* NO INPUT SAMPLEING */
    printf("INIT: NO INPUT TOGGLE, SUSPECT DISCONNECTED INPUT!\n");
    perform = FALSE;        /* INDICATE FAILURE */
    }
  perform = TRUE;         /*>>>>>>>>>>>>>>>>>>>>>>>>>>DUMMY SUCCESS */
  return(perform);          /* RETURN INDICATOR */
  }

/*********************************/
/* PICK_IT_OUT()                 */
/*                               */
/* OPTION SELECTION ROUTINE      */
/*********************************/
int  pick_it_out()
  {
  int  selection;                   /*RESPONSE HOLDER*/

  side_show();                      /*PUT UP A MENU*/
  selection = getch();              /*MAKE RESPONSE*/
  return(selection);                /*RETURN SELECTION*/
  }

/*********************************/
/* SIDE_SHOW()                   */
/*                               */
/* MENU GENERATION ROUTINE       */
/*********************************/
side_show()
  {
  printf("\nSelect LETTER for the function desired!\n\n");
  printf("\tA  => ANALYZE DATA!\n");
  printf("\tD  => DISPLAY DATA!\n");
  printf("\tF  => FILE DATA!\n");
  printf("\tG  => GATHER DATA!\n");
  printf("\tP  => POLE I/O LOCATIONS!\n");
  printf("\tS  => SWEEP I/O LOCATIONS!\n");
  printf("\tX  => Exit program\n\n");
  }

/************************************/
/* POLE_IT()                        */
/*                                  */
```

```
 unsigned int perform;
 unsigned int holder;

 while(kbhit()==FALSE)
   {
   holder = INMASK;
   holder &= inp(STATUS);

   switch(holder)
     {
     default:
/*       printf("Status -> %x\n",holder);*/
       break;
     }
   }
  perform = TRUE;
  return(perform);
  }

/************************************/
/* SHOW_IT()                        */
/* Gather and display data.         */
/************************************/
int show_it()
  {
  int perform;                      /* PERFORMANCE INDICATOR */
  unsigned int status;              /* MANIPULATION VARIABLE */
  unsigned int holder;              /* MANIPULATION VARIABLE */
  unsigned int timeout;             /* TIMER VARIABLE */
  unsigned int scan;                /* COUNTER VARIABLE */

  status = inp(B_STAT);             /* SET UP FOR SYNC DISCRIMINATION */
  while((status & SYNC) == SYNC)    /* RESET FIFOS AND LOOP */
    {
    outp(RESET,DUMMY);              /* RESET THE FIFOS */
    status = inp(B_STAT);           /* SET UP FOR SYNC DISCRIMINATION */
    }
  while((status & SYNC) == 0)       /* RESET FIFOS AND LOOP */
    {
    outp(RESET,DUMMY);              /* RESET THE FIFOS */
    status = inp(B_STAT);           /* SET UP FOR SYNC DISCRIMINATION */
    }
  for(scan=0;scan<SAMPLE_PTS;scan++)/* GET EACH SCAN */
    {
    status = inp(STATUS) & HLFE;    /* HIGH PERIOD LOW BYTE CHECK */
    for(timeout=T_BTWN_SCANS;(timeout>0)&&(status==0);timeout--) /* TIME BETWEEN
      {                                                          /* WAIT FOR COR
      status = inp(STATUS)& HLFE;   /* GET AND MASK STATUS */
      }

    if(timeout == 0)                /* INDICATE TIMEOUT ERROR */
      {
      printf("Timeout on HIGH PERIOD read!SCAN->%d\n",scan);
      }
    else                            /* DATA IS READY, GET IT */
      {
      period_hi[scan] = inp(R_HH) << 8;    /* GET THE HIGH BYTE AND PLACE IT*/
      period_hi[scan] |= inp(R_HL); /* MASK IN LOWER BYTE */
      }

    status = inp(STATUS) & LLFE;    /* LOW PERIOD LOW BYTE CHECK */
    for(timeout=T_BTWN_SCANS;(timeout>0)&&(status==0);timeout--) /* TIME BETWEEN
```

```
    if(timeout == 0)                /* INDICATE TIMEOUT ERROR */
      {
      printf("Timeout on LOW PERIOD read!\n");
      }
    else                            /* DATA IS READY, GET IT */
      {
      period_lo[scan] = inp(R_LH) << 8;    /* GET AND PLACE THE HIGH BYTE */
      period_lo[scan] |= inp(R_LL); /* MASK IN LOWER BYTE */
      }
    }
  data_taken = TRUE;
  printf("Scan Complete!\n");
  }


/************************************/
/* SWEEP_IT()                       */
/*                                  */
/* CONTINUOUSLY BEAT ON I/O's       */
/************************************/
int  sweep_it()
  {
  unsigned int perform;
  unsigned int holder;

  holder = 0X300;

  while(kbhit()==FALSE)
    {
    for(holder = 0x300;holder<0x308;holder++)
      {
      perform =  inp(holder);
      }
    }
  perform = TRUE;
  return(perform);
  }

/************************************/
/* DISPLAY_IT()                     */
/*                                  */
/* Show the data!                   */
/************************************/
int  display_it()
  {
  unsigned int perform;
  unsigned int scan;

  for(scan = 0;scan < 1024;scan++)
    {
    printf("period %d low->%x\n",scan,period_lo[scan]);
    printf("period %d hi->%x\n",scan,period_hi[scan]);
    }
  perform = TRUE;
  return(perform);
  }

/************************************/
/* FILE_IT()                        */
/*                                  */
/* File the data!                   */
/************************************/
```

```
unsigned int count;
int radix;
char buffer[20];
char buffer_lo[7];
char buffer_hi[7];
char *to_here;

if(analyze_it()==TRUE)
  {
  if((hi_lo = fopen("a:info.dat","w"))== NULL)
    {
    printf("FILE_IT:File open error!\n");
    perform = FALSE;
    }
  else
    {
    radix = 10;
    count = sprintf(buffer,"%d\n",max_samnum_lo);
    count = fwrite(buffer,1,6,hi_lo);                    /* WRITE TO THE FILE */

    for(scan = 1;scan < max_samnum_lo;scan++)
      {
      count = sprintf(buffer_lo,"%d",period_lo[scan]);
      count = sprintf(buffer_hi,"%d\n",period_hi[scan]);

      for(count = 0;count < 5;count++)
        {
        buffer[count] = buffer_lo[count];
        }
      for(count = 5;count < 10;count++)
        {
        buffer[count] = 0x20;
        }
      for(count = 0;count < 7;count++)
        {
        buffer[count + 10] = buffer_hi[count];
        }
      buffer[15] = 0x0d;
      buffer[16] = 0x0a;
      count = fwrite(buffer,1,17,hi_lo);                 /* WRITE TO THE FILE */
      }
    }
   if(hi_lo == NULL)
     {
     printf("File never opened!\n");
     }
   else
     {
     fclose(hi_lo);
     printf("FILE CLOSED!\n");
     }
   perform = TRUE;
   }
  else
    {
    printf("FILE_IT:Analysis failure!\n");
    perform = FALSE;
    }
  return(perform);
  }

/***********************************/
```

```
int  analyze_it()
  {
  unsigned int perform;
  unsigned int scan;
  unsigned int lockout_hi;
  unsigned int lockout_lo;
  unsigned int min_sampl_hi;
  unsigned int max_sampl_hi;
  unsigned int min_sampl_lo;
  unsigned int max_sampl_lo;
  unsigned int max_scan_hi;
  unsigned int max_scan_lo;
  unsigned long mean_hi;
  unsigned long mean_lo;
  unsigned long runsum_hi;
  unsigned long runsum_lo;

  if(data_taken == TRUE)
    {
    max_samnum_hi = 0;
    lockout_hi = FALSE;

    max_sampl_hi = period_hi[0];
    min_sampl_hi = period_hi[0];
    runsum_hi = (unsigned long)period_hi[0];

    for(scan = 1;(scan < 1024)&&(lockout_hi==FALSE);scan++)
      {
      runsum_hi += (unsigned long)period_hi[scan];
      if((period_hi[scan] < min_sampl_hi)&&(period_hi[scan] != 0))
        {
        min_sampl_hi = period_hi[scan];
        }
      else
        {
        if(period_hi[scan] == 0)
          {
          max_samnum_hi = scan;
          lockout_hi = TRUE;              /* BLOCK OUT FURTHER SCANS */
          }
        else
          {
          if(period_hi[scan] > max_sampl_hi)
            {
            max_sampl_hi = period_hi[scan];
            max_scan_hi = scan;
            }
          }
        }
      }
    if(scan == 1024)
      {
      max_samnum_hi = 1024;
      }
    max_samnum_lo = 0;
    lockout_lo = FALSE;

    max_sampl_lo = period_lo[0];
    min_sampl_lo = period_lo[0];
    runsum_lo = (unsigned long)period_lo[0];

    for(scan = 1;(scan < 1024)&&(lockout_lo==FALSE,.scan++)
```

```
        min_sampl_lo = period_lo[scan];
        }
      else
        {
        if(period_lo[scan] == 0)
          {
          max_samnum_lo = scan;
          lockout_lo = TRUE;              /* BLOCK OUT FURTHER SCANS */
          }
        else
          {
          if(period_lo[scan] > max_sampl_lo)
            {
            max_sampl_lo = period_lo[scan];
            max_scan_lo = scan;
            }
          }
        }
      }
    if(scan == 1024)
      {
      max_samnum_lo = 1024;
      }
    mean_hi = runsum_hi/(unsigned long)max_samnum_hi;
    mean_lo = runsum_lo/(unsigned long)max_samnum_lo;

    printf("Maximum sample LOW = %x at %d\n",max_sampl_lo,max_scan_lo);
    printf("Minimum sample LOW = %x\n",min_sampl_lo);
    printf("Maximum sample HI = %x at %d\n",max_sampl_hi,max_scan_hi);
    printf("Minimum sample HI = %x\n\n",min_sampl_hi);
    printf("Maximum number of HI samples->%d\n",max_samnum_hi);
    printf("Maximum number of LO samples->%d\n",max_samnum_lo);
    printf("MEAN of HI samples->%ld\n",mean_hi);
    printf("MEAN of LO samples->%ld\n",mean_lo);
    perform = TRUE;
    }
  else
    {
    printf("No DATA TAKEN!\n");
    perform = FALSE;
    }

  return(perform);
  }

/***********************************/
/*EXECUTE(FEATURE)                 */
/*EXECUTE SELECTED FEATURE         */
/***********************************/
int  execute(feature)
char  feature;
  {/*PERFORM FEATURE DESIRED*/
  int  validity;  /*VALIDITY INDICATOR*/

  validity = TRUE;  /*DEFAULT TO TRUE*/

  switch(feature)
    {/*PICK AND PERFORM*/

    case 'A':
    /* DISPLAY DATA */
      validity = analyze_it();       /* SEPARATE DATA INTO CATAGORIES */
```

```
    validity = display_it();        /* GRAB AND DISPLAY */
    break;

  case 'F':
  /* FILE THE DATA */
    validity = file_it();           /* FILE THE DATA */
    break;

  case 'G':
  /* GATHER AND DISPLAY DATA */
    validity = show_it();           /* GRAB AND DISPLAY */
    break;

  case 'P':
  /* POLE THE I/O LOCATION */
    validity = pole_it();           /* CONSTANTLY CHECK STATUS */
    break;

  case 'S':
  /* SWEEP THE I/O LOCATION */
    validity = sweep_it();          /* SWEEP THROUGH THE I/O LOCATIONS */
    break;

  case 'X':
    printf("Say Bye!\n");
    validity = TRUE;
    break;


  default:
    /*INVALID SELECTION*/
    validity = FALSE;
    break;
  }
return(validity);
}
```

```
/*************************************/
/* Gate Board Definition Header File */
/*************************************/

#define  TRUE         0x01              /* ASSIGN TRUE A VALUE          */
#define  FALSE        0x00              /* ASSIGN FALSE A VALUE          */

#define  ON           0x01              /* On value  (Logic True)       */
#define  OFF          0x00              /* Off value (Logic False)       */
#define  INMASK       0xFF              /* NO MASKING */

#define  DEF_ADDR     0x300             /* Default Base IO Address */
#define  DUMMY        0X00              /* DUMMY VALUE */
#define  T_BTWN_SCANS 0x40              /* COUNTS BETWEEN SCANS */
#define  SAMPLE_PTS   1024              /* 1024 SAMPLE POINTS */

/* I/O LOCATIONS */
#define  R_LL         DEF_ADDR          /* READ LOWER BYTE LOW SIDE */
#define  R_LH         DEF_ADDR + 1      /* READ HIGH BYTE LOW SIDE */
#define  R_HL         DEF_ADDR + 2      /* READ LOW BYTE HIGH SIDE */
#define  R_HH         DEF_ADDR + 3      /* READ HIGH BYTE HIGH SIDE */
#define  RESET        DEF_ADDR + 4      /* RESET ALL FIFOS */
#define  STATUS       DEF_ADDR + 5      /* FIFO STATUS */
#define  B_STAT       DEF_ADDR + 6      /* BOARD STATUS */

/* BIT DEFINES */
#define  LLFF        0x01              /* LOW BYTE LOW SIDE FIFO FULL */
#define  LLFE        0x10              /* LOW BYTE LOW SIDE FIFO EMPTY */
#define  LHFF        0x02              /* HIGH BYTE LOW SIDE FIFO FULL */
#define  LHFE        0x20              /* HIGH BYTE LOW SIDE FIFO EMPTY */
#define  HLFF        0x04              /* LOW BYTE LOW SIDE FIFO FULL */
#define  HLFE        0x40              /* LOW BYTE LOW SIDE FIFO EMPTY */
#define  HHFF        0x08              /* HIGH BYTE LOW SIDE FIFO FULL */
#define  HHFE        0x80              /* HIGH BYTE LOW SIDE FIFO EMPTY */
#define  SYNC        0x01              /* SYNC ACTIVE INDICATOR */
```

```
TITLE           GATEBOARD Interface pal
PATTERN
REVISION        001
AUTHOR          C.T. (TOMCAT) Cathey
COMPANY         Tri Square Technical Services
DATE            21 MARCH 1991
;
CHIP            STD_IO  PAL16L8

;PINS   1       2       3       4       5       6       7       8       9
        /SEL    A0      A1      A2      A3      A4      PIN7    PIN8    PIN9

;PINS   11      12      13      14      15      16      17      18      19
        PIN11   /RDLOL  /RDLOU  /RDHIL  /RDHIU  /STATUS /RSET   PIN18   /DATA

EQUATIONS

;       /RDLOL READS THE LO COUNT LSB
RDLOL   = SEL * /A0 * /A1 * /A2 * /A3
RDLOL.TRST = VCC

;       /RDLOU READS THE LO COUNT MSB
RDLOU   = SEL * A0 * /A1 * /A2 * /A3
RDLOU.TRST = VCC

;       /RDHIL READS THE HI COUNT LSB
RDHIL   = SEL * /A0 * A1 * /A2 * /A3
RDHIL.TRST = VCC

;       /RDHIU READS THE HI COUNT MSB
RDHIU   = SEL * A0 * A1 * /A2 * /A3
RDHIU.TRST = VCC

;       /RSET RESETS ALL THE FIFOS
RSET    = SEL * /A0 * /A1 * A2 * /A3
RSET.TRST = VCC

;       /DATA ENABLES COMMUNICATION WITH FIFOS
DATA = SEL * /A3
DATA.TRST = VCC

;       /STATUS ENABLES STATUS CHECK OF FIFOS
STATUS = SEL * A0 * /A1 * A2 * /A3
STATUS.TRST = VCC

SIMULATION

TRACE_ON A0 A1 A2 A3 SEL RDLOL RDLOU RDHIL RDHIU RSET
         DATA STATUS

SETF /A0 /A1 /A2 /A3 SEL
SETF A0 /SEL
SETF SEL
SETF A1 /A0 /SEL
SETF SEL
SETF A1 A0 /SEL
SETF SEL
SETF A2 /A1 /A0 /SEL
SETF SEL
SETF A2 /A1 A0 /SEL
SETF SEL
```

```
SETF A3 /A2 /A1 /A0 /SEL
SETF SEL
SETF A0 /SEL
SETF SEL
SETF A1 /A0 /SEL
SETF SEL
SETF A1 A0 /SEL
SETF SEL
SETF A2 /A1 /A0 /SEL
SETF SEL
SETF A2 /A1 A0 /SEL
SETF SEL
SETF A2 A1 /A0 /SEL
SETF SEL
SETF A2 A1 A0 /SEL
SETF SEL
SETF /A3 /A2 /A1 /A0 /SEL
SETF SEL

TRACE_OFF
```

Blank

APPENDIX B
PROGRAM LISTING 2. TICOUNT.BAS

```
100 DIM HC(2048),LC(2048)
102 PRINT 'Enter an option '
105 PRINT ' 1 initializes the FIFOs'
110 PRINT ' 2 reads the FIFOs'
120 PRINT ' 3 saves single scan period data'
130 PRINT ' 4 analyzes single scan data '
140 PRINT ' 5 saves residuals for single scan'
150 PRINT ' 6 loads single scan period data'
190 PRINT ' 7 exits program'
200 INPUT ' Enter selection ';R
210 ON R GOSUB 9000,2000,3000,5000,6000,7000,240
220 CLS
230 GOTO 105
240 END
```

```
2000 '*****************************************************
2010 '   Read data from FIFOs and store it in high/low count
2020 '   arrays.
2030 '*****************************************************
2050 CLS
2060 FOR I=0 TO 3: PRINT INP(768+I);:NEXT I
2070 SW=1: I=0
2080 WHILE SW
2090 HHC=INP(769):HLC=INP(768)
2100 HC(I)=HHC*256+HLC
2110 LHC=INP(771):LLC=INP(770)
2120 LC(I)=LHC*256+LLC
2130 I=I+1
2140 ST=INP(773)
2150 IF ST=15 THEN SW=0
2160 WEND
2170 T=I-1
2180 PRINT 'Terminal value ';T
2190 RETURN
3000 '*****************************************************
3010 '   Save raw data from a single scan  of data
3020 '   for calculation of velocity variation in the scan.
3030 '*****************************************************
3040 OPEN 'o',#1,'b:ssv.dat'
3050 PRINT #1,T
3060 FOR I=0 TO T
3070 PRINT #1,LC(I);',';HC(I)
3080 NEXT I
3090 CLOSE #1
3100 RETURN
```

```
5000 '*****************************************************
5010 '  Analyze a single scan for variation in the mirror
5020 '  velocity.
5030 '*****************************************************
5040 '*****************************************************
5050 '  Find the minimum and maximum values in the periods
5060 '*****************************************************
5070 HMAX=HC(0): HMIN=HC(0): JMAX=0: JMIN=0
5080 FOR I=0 TO T
5090 IF HMAX<HC(I) THEN JMAX=I: HMAX=HC(I)
5100 IF HMIN>HC(I) THEN JMIN=I: HMIN=HC(I)
5110 NEXT I
5120 LMAX=LC(0): LMIN=LC(0): KMAX=0: KMIN=0
5130 FOR I=0 TO T
5140 IF LMAX<LC(I) THEN KMAX=I: LMAX=LC(I)
5150 IF LMIN>LC(I) THEN KMIN=I: LMIN=LC(I)
5160 NEXT I
5170 TMAX=HMAX: TMIN=HMIN
5180 IF LMAX>TMAX THEN TMAX=LMAX
5190 IF LMIN<TMIN THEN TMIN=LMIN
5200 '*****************************************************
5210 '  Determine the distribution of values in the
5220 '  high and low periods of the data set
5230 '*****************************************************
5240 IL=INT(LMAX-LMIN): IH=INT(HMAX-HMIN)
5250 DIM HD(IH),LD(IL),HDC(IH),LDC(IL)
5260 FOR I=0 TO IH: HD(I)=HMIN+I: NEXT I
5270 FOR I=0 TO IL: LD(I)=LMIN+I: NEXT I
5280 FOR P=0 TO IH
5290 FOR I=0 TO T
5300 IF HC(I)=HD(P) THEN HDC(P)=HDC(P)+1
5310 NEXT I
5320 NEXT P
5330 FOR P=0 TO IL
5340 FOR I=0 TO T
5350 IF LC(I)=LD(P) THEN LDC(P)=LDC(P)+1
5360 NEXT I
5370 NEXT P
5380 '*****************************************************
5390 '  Calculate the average value and the best estimate of
5400 '  the standard deviation
5410 '*****************************************************
5420 HT=0: LT=0: HP=0: LP=0: HR=0: LR=0: TR=0
5430 FOR I=0 TO IH: HP=HDC(I)+HP: HT=HD(I)*HDC(I)+HT: NEXT I
5440 FOR I=0 TO IL: LP=LDC(I)+LP: LT=LD(I)*LDC(I)+LT: NEXT I
5450 AT=HT/HP: AL=LT/LP: AV=(AT+AL)/2
5460 FOR I=0 TO IH: HR=(HD(I)-AT)*(HD(I)-AT)*HDC(I)+HR: NEXT I
5470 DH=SQR(HR/(HP-1))
5480 FOR I=0 TO IL: LR=(LD(I)-AL)*(LD(I)-AL)*LDC(I)+LR: NEXT I
5490 DL=SQR(LR/(LP-1))
5500 FOR I=0 TO IH: TR=(HD(I)-AV)*(HD(I)-AV)*HDC(I)+TR: NEXT I
5510 FOR I=0 TO IL: TR=(LD(I)-AV)*(LD(I)-AV)*LDC(I)+TR: NEXT I
5520 DT=SQR(TR/(LP+HP-1))
5530 '*****************************************************
5540 '  Display results of analysis
5550 '*****************************************************
5560 PRINT 'Average  High period value ';AT;' with BESD of ';DH
5570 PRINT 'Average  Low period value ';AL;' with BESD of ';DL
5580 PRINT 'Average  Overall value ';AV;' with BESD of ';DT
5590 PRINT ' Distribution of High Period Values '
5600 FOR I=0 TO IH: PRINT 'Hi pd val ';HD(I);' X ';HDC(I): NEXT I
5610 PRINT ' Distribution of Low Period Values '
5620 FOR I=0 TO IL: PRINT 'Lo pd val ';LD(I);' X ';LDC(I): NEXT I
5630 V$=INKEY$: IF V$()'c' THEN GOTO 5630
5640 RETURN
```

```
6000 '**********************************************************
6010 '  Save residuals for a single interferometer mirror scan
6020 '**********************************************************
6030 OPEN 'o',#1,'b:ssvres.dat'
6040 PRINT #1,T
6050 FOR I=0 TO T
6060 PRINT #1, LC(I)-AV;' , ';HC(I)-AV
6070 NEXT I
6080 CLOSE #1
6090 RETURN
7000 '**********************************************************
7010 '  Read in a sequential file containing period counts
7020 '**********************************************************
7030 OPEN 'i',#1,'b:ssv.dat'
7040 INPUT #1,T
7050 FOR I=0 TO T
7060 INPUT #1,LC(I),HC(I)
7070 NEXT I
7080 RETURN
9000 '**********************************************************
9010 '  Initialize all FIFOs
9020 '**********************************************************
9030 RST=INP(772): ST=INP(773)
9040 PRINT 'Status word ';ST
9050 RETURN
```

```
B:\>type ssv.dat
7
55 , 55
54 , 54
51 , 51
55 , 55
53 , 53
53 , 53
54 , 54
52 , 52

B:\>
```

File format:
number of values -1
low value, high value
low value, high value
. .
. .
. .
last low value, last high value

Screen #1: Menu

```
run
Enter an option*
 1 initializes the FIFOs
 2 reads the FIFOs
 3 saves single scan period data
 4 analyzes single scan data
 5 saves residuals for single scan
 6 loads single scan period data
 7 exits program
 Enter selection ? 6
```

Screen #2: Menu with results of test file SSV.DAT

```
 1 initializes the FIFOs
 2 reads the FIFOs
 3 saves single scan period data
 4 analyzes single scan data
 5 saves residuals for single scan
 6 loads single scan period data
 7 exits program
 Enter selection ? 4
Average  High period value  53.375  with BESD of  1.407886
Average  Low period value  53.375  with BESD of  1.407886
Average  Overall value  53.375  with BESD of  1.360147
 Distribution of High Period Values
Hi pd val  51  X  1
Hi pd val  52  X  1
Hi pd val  53  X  2
Hi pd val  54  X  2
Hi pd val  55  X  2
 Distribution of Low Period Values
Lo pd val  51  X  1
Lo pd val  52  X  1
Lo pd val  53  X  2
Lo pd val  54  X  2
Lo pd val  55  X  2
```